ELSEVIER

Theoretical Computer Science 240 (2000) 257–269

Theoretical Computer Science

www.elsevier.com/locate/tcs

for NL-hard sparse sets

Jin-Yi Cai[a,1], D. Sivakumar[b,*,2]

[a] *Department of Computer Science, State University of New York, Buffalo, NY 14260, USA*
[b] *Department of Computer Science, University of Houston, Houston, TX 77204-3475, USA*

Dedicated to the memory of J. Myhill

Abstract

We resolve a conjecture of Hartmanis from 1978 about sparse hard sets for nondeterministic logspace (NL). We show that there exists a sparse hard set S for NL under logspace many-one reductions if and only if NL = L (deterministic logspace). © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Sparse sets; Isomorphism conjecture; Nondeterministic space; Circuit complexity; Algebraic computation

1. Introduction

A set is sparse if it has at most a polynomial number of strings of each length n. Sparse hard sets (and sparse complete sets) have been a fascinating subject of study in complexity theory for the past two decades. The research on sparse hard sets for complexity classes has two progenitors: the isomorphism theorems and conjectures, and the connection with circuit complexity.

The roots of the research reported here can be traced back to the work of logicians on isomorphism theorems, especially to the isomorphism theorem of the late logician, our colleague at SUNY Buffalo, J. Myhill, to whom we dedicate this work. The Myhill isomorphism theorem (see [19]) states that all r.e. complete languages are isomorphic

* Corresponding author.
E-mail addresses: cai@cs.buffalo.edu (J.-Y. Cai), siva@cs.uh.edu (D. Sivakumar).

[1] Research supported in part by NSF grants CCR-9057486 and CCR-9319093, and by an Alfred P. Sloan Fellowship.
[2] Most of this research was done while the author was at SUNY/Buffalo, supported in part by NSF grant CCR-9409104.

PII: S0304-3975(99)00234-0

under recursive 1–1 reductions. The elegance of this theorem is its unifying conceptual simplicity — up to recursive permutations, there is just one r.e. complete language, the halting problem of Turing.

In the 1970s Berman and Hartmanis [1] showed that all known NP-complete languages are isomorphic under polynomial-time computable reductions. Moreover, Hartmanis [8] also showed that the same is true for P and NL, namely that all known P-complete sets, and NL-complete sets, respectively, are isomorphic to each other under bijections computable (and invertible) in logspace. Following this evidence, and the similarity with Myhill's theorem, Berman and Hartmanis [1], and Hartmanis [8], respectively, conjectured that all such complete languages within the respective complexity classes are indeed isomorphic to each other. These isomorphism conjectures appear far from being resolved.

As consequences of the isomorphism conjectures, using a density argument, they also conjectured that no sparse set can be complete within the respective complexity classes NP,P, and NL, unless the various complexity classes collapse. The subject of the present paper is sparse hard sets for the complexity class NL, nondeterministic logspace.

Conjecture 1 (Hartmanis [8]). *NL has a sparse complete set under logspace many-one reductions iff* $\mathrm{NL} = \mathrm{L}$.

Clearly, if $\mathrm{NL} = \mathrm{L}$ then sparse complete sets exist for NL. The sparse set conjecture states that if $\mathrm{NL} \neq \mathrm{L}$ then there is no NL-complete sparse set.

There is a second link of sparse sets with complexity via circuit complexity theory. A fundamental result of A. Meyer [1] states that a language has a polynomial size circuit family if and only if it can be reduced via Cook reductions (polynomial time Turing reductions) to a sparse set. As a consequence, every NP language has a polynomial size circuit family if and only if some NP-complete language can be reduced by Cook reductions to a sparse set. The well-known Karp–Lipton theorem states that if a sparse hard set exists for NP under Cook reductions, then the polynomial-time hierarchy collapses to its second level Σ_2^p [12]. For the history and survey of interesting developments concerning sparse sets, see the articles [9, 22, 23, 3].

We note here (see also [4]) that the connection between sparse sets and circuit complexity extends to low-level circuit complexity as well. It can be shown that a language has constant-depth, polynomial-size circuits that use AND, OR, NOT, and MAJORITY gates iff it is reducible to a sparse set via reductions computable by a uniform family of such circuits. In other words, a language A belongs to nonuniform TC^0 iff A is reducible to a sparse set S via logspace-uniform TC^0 reductions that make at most polynomial number of queries to the sparse set S. Similarly, a language A belongs to nonuniform NC^1 iff A is reducible to a sparse set S via logspace-uniform NC^1 reductions that make at most polynomial number of queries to the sparse set S. (For exact definitions of TC^0, NC^1, and of logspace uniformity, see Section 2.) The

class TC^0 is one of the "weakest" complexity classes whose limitations are still far from being understood. In particular, it is still open if nonuniform TC^0 is different from deterministic exponential time; it is also open whether logspace-uniform TC^0 is different from any of logspace-uniform NC^1, Logspace, NL, or even from P.

The Berman–Hartmanis conjecture concerning NP-complete sparse sets was settled by Mahaney in 1980 [13]. He showed that NP has sparse hard sets under polynomial-time many-one reductions iff $NP = P$. The techniques used by Mahaney and subsequently built upon by Ogiwara and Watanabe [16] do not appear to work for the conjectures concerning sparse hard sets for P and for NL. Building on significant progress by Ogihara [15], the conjecture for P was settled by Cai and Sivakumar [4]. It was shown in [4] that P has sparse hard sets under logspace many-one reductions iff $P = L$.

In this paper, we finally settle the Hartmanis conjecture for NL. We show that there is a sparse hard set for NL under logspace many-one reductions iff $NL = L$. Our proof builds on [15, 4] and uses the algebraic techniques of Cai and Sivakumar [4]. An additional crucial ingredient in the proof is the famous result of Immerman [11] and Szelepcsényi [20], which shows $NL = co\text{-}NL$. Previously, Cai et al. [2] built on the ideas of Cai and Sivakumar [4] and showed that if there is a sparse set S that is hard for NL under logspace many-one reductions, then every NL problem can be solved by a probabilistic logspace machine that is given two-way access to a random tape.

Assuming the existence of a sparse hard set for NL, our proof gives a parallel algorithm for an NL-complete problem. This parallel algorithm can be implemented by a family of logspace-uniform circuits of polynomial size and constant depth built using AND, OR, NOT, and MAJORITY gates, with polynomially many parallel calls to the reduction from NL to the sparse set S. This implies that if NL has a sparse hard set under logspace many-one reductions, then $NL = L$, and if NL has a sparse hard set under logspace-uniform TC^0 many-one reductions, then $NL =$ logspace-uniform TC^0.

Based on a preliminary draft of this paper, van Melkebeek [14] has extended the result to the case of sparse hard sets for NL under bounded truth-table reductions (that is, reductions that make a constant number of queries).

2. Preliminaries

All our notations and definitions are standard. We denote by P the class of all languages recognizable in polynomial time by deterministic Turing machines. The class of all languages recognizable by deterministic Turing machines that use space no more than $O(\log n)$ is denoted Logspace or L; the corresponding nondeterministic class is denoted by NL.

For circuit and parallel complexity, we adopt the following notion of *uniformity*: a circuit family $\{C_n\}_{n=0}^{\infty}$, where C_n is of size $s(n)$ is said to be *logspace uniform* if there is a deterministic space $(\log s(n))$-bounded transducer that, on input 0^n, outputs an encoding of the circuit C_n. (There are much finer notions of uniformity, but we will not use them in this paper.) The class AC^0 consists of languages accepted by

a logspace-uniform family of constant-depth, polynomial-size circuits that use unary NOT gates and AND and OR gates of unbounded fanin. The class TC^0 consists of languages accepted by a logspace-uniform family of constant-depth, polynomial-size circuits that use unary NOT gates and AND, OR, and MAJORITY gates of unbounded fanin. (A MAJORITY gate with k inputs outputs 1 iff $\geqslant k/2$ inputs are 1.) The class NC^1 consists of languages accepted by a logspace-uniform family of logarithmic-depth, polynomial-size circuits that use unary NOT gates and AND and OR gates of bounded fanin. The following well-known relations hold among these complexity classes: $AC^0 \subseteq TC^0 \subseteq NC^1 \subseteq L \subseteq NL$.

As a minor abuse of notation, we often write "AC^0 circuit" to mean a constant depth polynomial size circuit that uses NOT gates and unbounded-fanin AND and OR gates; and we write "TC^0 circuit" to mean a constant depth polynomial size circuit that uses NOT gates and unbounded-fanin AND, OR, and MAJORITY gates.

For any language A, let $c_A(n) \doteq \|\{x \in A \mid |x| \leqslant n\}\|$ denote the *census function* for A. A is called (polynomially) *sparse* if $c_A(n)$ is bounded by a polynomial in n.

Given a directed graph $G=(V,E)$ and two distinguished vertices $s,t \in V$, the *s–t connectivity problem* asks whether there is a directed path from s to t in G, i.e., whether a sequence of directed edges $(s,u_1),(u_1,u_2),\ldots,(u_k,t)$ exists. The *s–t* connectivity problem is well-known to be complete for NL under logspace many-one reductions [18]. Immerman [11] has shown that this problem is complete for NL under an extremely weak form of many-one reductions called first-order projections (that are, in fact, quantifier-free). We note that the *s–t* connectivity problem remains NL-complete even when restricted to directed acyclic graphs. We call this problem *DAG-STCON*. Moreover, without loss of generality, we may also assume that all instances of *DAG-STCON* are labeled and layered graphs, that is, graphs where all edges go from lower-numbered vertices to higher-numbered vertices. The completeness of *DAG-STCON* for NL implies that if *DAG-STCON* $\in TC^0$, then $NL = TC^0$; if *DAG-STCON* $\in NC^1$, then $NL = NC^1$; and if *DAG-STCON* $\in L$, then $NL = L$.

3. Main result

Theorem 1. *If there is a sparse set S that is hard for NL under logspace many-one reductions, then DAG-STCON can be solved by a logspace-uniform family of constant-depth, polynomial-size circuits that use unary NOT gates and AND, OR, and MAJORITY gates of unbounded fan-in, and that make polynomially many parallel queries to the reduction from NL to S.*

That is, modulo the complexity of the reduction, the parallel algorithm for *DAG-STCON* works in TC^0. It follows that if there is a sparse hard set for NL under many-one reductions computable in TC^0, then NL equals logspace-uniform TC^0, and that if there is a sparse hard set for NL under many-one reductions computable in logspace, then $NL = L$.

The rest of this section is devoted to the proof of Theorem 1. We begin with an outline of the sequence of arguments employed.

(1) We will first define a witness function W for instances of *DAG-STCON*, prove its correctness, and point out some of its nice properties.
(2) Using the witness function W, we will define a language B that will be an extension of *DAG-STCON*. Using properties of W and some finite field computations, we will present a nondeterministic logspace algorithm for B to establish $B \in \mathrm{NL}$.
(3) We will then invoke the hypothesis, and describe the TC^0 algorithm for *DAG-STCON*, which has the following structure:
(3a) On input $\langle G,s,t\rangle$, first produce polynomially many systems of linear equations whose unknowns are the bits of the witness function W applied to $\langle G,s,t\rangle$, and whose coefficient matrices are Vandermonde matrices. This step makes crucial use of the reduction from B to the sparse set S, and guarantees that at least one of the systems of equations is correct with respect to the bits of $W(G,s,t)$.
(3b) Using TC^0 computations, solve the systems of linear equations produced. The key ingredients of this step are the Lagrange interpolation formula and the Discrete Fourier Transform in finite fields.
(3c) Verify the correctness of the solution of every system of equations to weed out all incorrect solutions. Using the correct solution (which is guaranteed to exist in Step (3a)), decide whether $\langle G,s,t\rangle \in DAG\text{-}STCON$.

3.1. The witness function W

Let $\langle G,s,t\rangle$ be an instance of *DAG-STCON*; let $V = V(G)$, $E = E(G)$, and $n = |V|$. Let $A = A(G)$ denote the adjacency matrix of G; by our assumptions about the instances of *DAG-STCON* (see Section 2), A is a strictly upper-triangular matrix. The witness $W = W(G,s,t)$ for the instance $\langle G,s,t\rangle$ is an $n \times n$ strictly upper-triangular 0–1 (Boolean) matrix indexed by pairs of vertices of G, where $W_{uv} = 1$ iff there is a path from u to t in G whose first step is the edge (u,v).

We point out the following properties of W:

(1) For all $u,v \in V$, $W_{uv} \leqslant A_{uv}$.
(2) For any instance $\langle G,s,t\rangle$ of *DAG-STCON*, W is *uniquely* defined.
(3) For any instance $\langle G,s,t\rangle$ of *DAG-STCON*, $\langle G,s,t\rangle \in DAG\text{-}STCON$ if and only if $[\bigvee_{v\in V} W_{sv}]$ is true.
(4) Every bit of W is an NL predicate. Formally, the language Z that consists of the tuples $\langle G,s,t,u,v,b\rangle$ where $\langle G,s,t\rangle$ is a valid instance of *DAG-STCON* and $W(u,v) = b$, where $W = W(G,s,t)$, belongs to NL. This language is the union of the languages $Z_0 = \{\langle G,s,t,u,v,1\rangle \mid W(u,v) = 1\}$ and $Z_1 = \{\langle G,s,t,u,v,0\rangle \mid W(u,v) = 0\}$, which are easily seen, respectively, to be in NL and *co*-NL. Since *co*-NL = NL [11, 20], both Z_0 and Z_1 are in NL, and since NL is closed under unions, $Z \in \mathrm{NL}$.

A consequence of this fact is the following: The nondeterministic logspace machines for Z_0 and Z_1 can be used to build a nondeterministic logspace machine M_W that, given

$\langle G,s,t\rangle$ and $u,v\in V(G)$, computes W_{uv} in the following strong sense: every computation either outputs the correct value of W_{uv} or aborts in a "DON'T KNOW" state, and at least one computation is guaranteed to output the correct value of W_{uv}.

(5) There is a simple first-order formula to check the validity of a purported witness matrix W for any instance $\langle G,s,t\rangle$ of *DAG-STCON*:

$$\varphi(W)\equiv(\forall u,v\in V)\left[W_{uv}=1\Leftrightarrow\left(A_{uv}=1\wedge\left(\left(\bigvee_w W_{vw}\right)\vee v=t\right)\right)\right].$$

In circuit complexity terms, whether a given matrix W is the correct witness matrix for an instance of *DAG-STCON* can be checked by a logspace-uniform AC^0 circuit.

We now prove the correctness of the formula φ given above. By the correctness of the formula φ, we mean the following: $\varphi(W)$ is true if and only if W is the correct witness for the instance $\langle G,s,t\rangle$. In other words, $\varphi(W)$ is true if and only if for every $u,v\in V$, $W_{uv}=1$ iff there is a path from u to t whose first step is the edge (u,v). It is clear that if W is the correct witness for $\langle G,s,t\rangle$, then $\varphi(W)$ is true. In what follows, we prove the converse.

For convenience, we define for each $v\in V$, a boolean variable $W_v=_{\text{def}}\bigvee_w W_{vw}$, that is, W_v is the disjunction of all the W_{vw}'s. Assume $\varphi(W)$ is true. There are two cases to consider:

Case 1: Suppose that for some $u<v$, $W_{uv}=1$. We will construct a path P from u to t whose first step is the edge (u,v). Since $\varphi(W)$ is true and $W_{uv}=1$, $A_{uv}=1$; we add the edge (u,v) to P. Also since $\varphi(W)$ is true, $W_{uv}=1$ implies that either $v=t$ or $W_v=1$. If $v=t$, we are done. If $W_v=1$, then for some $w>v$, $W_{vw}=1$, and we will add the edge (v,w) to P.

If we continue this process for at least n steps, each step visiting a new vertex, one of two things must happen: either we reach t or we visit a vertex z that has been visited before. Since the latter possibility contradicts the assumption that G is acyclic, it must be the case that we visit t, in which case P constitutes a path from u to t whose first step is v.

Case 2: Suppose that for some u,v, there is a path Q from u to t whose first step is the edge (u,v). We will show that $W_{uv}=1$. Precisely, we will show that for every edge $(x,y)\in Q$, $W_{xy}=1$. For $(x,y)=(u,v)$, this gives $W_{uv}=1$. Let (w,t) be the last edge in the path Q. This implies that $A_{wt}=1$ and since $\varphi(W)$ is true, this implies that $W_{wt}=1$. Assume inductively that the assertion is true for some edge $(y,z)\in Q$, and let (x,y) be the edge preceding (y,z) in Q. Clearly $A_{xy}=1$; since $W_{yz}=1$ by the inductive hypothesis, the truth of $\varphi(W)$ implies that $W_{xy}=1$.

This completes the proof of the correctness of the formula φ for checking the validity of a purported witness W.

3.2. The language B

It is known that if m is of the form $2\cdot 3^{\ell}$ for some integer $\ell\geqslant 0$, the polynomial $x^m+x^{m/2}+1$ is an irreducible polynomial of degree m over $GF(2)$ [21]. In the

following, by a finite field $GF(2^m)$, where $m = 2 \cdot 3^\ell$, we refer explicitly to the field $\mathbf{Z}_2[x]/(x^m + x^{m/2} + 1)$. Following [15, 4], we will define an auxiliary language $B \in \mathrm{NL}$. Unlike the situation in these papers, the definition of B does not immediately imply that B belongs to NL; nevertheless, by taking advantage of the special properties of the witness function W and the ability to perform the necessary $GF(2^m)$ arithmetic in logspace, we are able to prove $B \in \mathrm{NL}$.

We define B to be the set of all tuples of the form $\langle G, s, t, 1^m, \alpha, \alpha_2, \ldots, \alpha_{k-1}, \beta\rangle$, where:

(1) $G = (V, E)$ is a directed, layered acyclic graph on n vertices, and has at most $k = \binom{n}{2}$ edges. Hence the adjacency matrix $A = A(G)$ of G is upper-triangular.
(2) s and t are vertices in G.
(3) m is of the form $2 \cdot 3^\ell$ for some integer $\ell \geqslant 0$.
(4) $\alpha, \beta \in GF(2^m)$.
(5) For $1 < i < k$, $\alpha_i = \alpha^i$ (that is, α_i is the ith power of α in $GF(2^m)$).
(6) $\sum_{i=0}^{k-1} \alpha^i W_{u_i v_i} = \beta$, where for $0 \leqslant i < k$, $0 \leqslant u_i < v_i < n$, (u_i, v_i) denotes the ith distinct vertex pair (edge/nonedge) in G, and $W = W(G, s, t)$.

This definition is somewhat complicated by the necessity of certain uniformity considerations. Intuitively, it appears that we should have been able to use just $\langle G, s, t, 1^m, \alpha, \beta\rangle$ in place of $\langle G, s, t, 1^m, \alpha, \alpha_2, \ldots, \alpha_{k-1}, \beta\rangle$, but unfortunately, this is not so simple. However, for the sake of readability, we will often abbreviate the notation $\langle G, s, t, 1^m, \alpha, \alpha_2, \ldots, \alpha_{k-1}, \beta\rangle$ by $\langle G, s, t, 1^m, \alpha, \beta\rangle$ throughout this paper.

We claim that the language B belongs to NL. We will build a nondeterministic logspace machine N that accepts B. It is clear that whether G is a directed layered acyclic graph can be verified using only $O(\log n)$ space. Next we argue that, in $O(\log n + \log m)$ space, N may deterministically verify that the values $\alpha_2, \alpha_3, \ldots, \alpha_{k-1}$ are indeed the correct powers of α. To do this, N proceeds sequentially, for i from 1 up to $k-2$, verifying the validity of $\alpha \cdot \alpha_i = \alpha_{i+1}$, where, for convenience, assume $\alpha_1 = \alpha$. For a fixed i, suppose that the correctness of $\alpha_j = \alpha^j$ has been verified for $j \leqslant i$. Now N sequentially computes each bit of the correct value of α^{i+1} from the values of α and $\alpha_i (= \alpha^i)$, and cross-checks it against α_{i+1}. The entire process requires two counters, one that can count up to $k-2$ and one that can count up to m. The counters can be implemented in space $O(\log n + \log m)$. Now to check the bits: For $\gamma \in GF(2^m)$, let $(\gamma)_j$ denote the jth bit of γ, and let $P_\gamma \in \mathbf{Z}_2[x]$ denote the polynomial of degree $< m$ whose coefficients are given by the bits of γ. Thus, $\alpha^{i+1} = (P_\alpha \cdot P_{\alpha^i}) \bmod (x^m + x^{m/2} + 1)$, where P_α and P_{α^i} are multiplied in the ring $\mathbf{Z}_2[x]$. For $0 \leqslant j \leqslant 2m-2$, the coefficient of x^j in $(P_\alpha \cdot P_{\alpha^i})$ is given by $\sum_{\sigma+\tau=j} (\alpha)_\sigma (\alpha^i)_\tau$. This is a mod 2 sum of at most m bits. Denote this sum by $S(j)$. When the product $(P_\alpha \cdot P_{\alpha^i})$ is reduced modulo $x^m + x^{m/2} + 1$, the jth bit of α^{i+1}, for $0 \leqslant j < m$, is given by the sum, in $\mathbf{Z}_2$, of the following four contributions $S_1(j), S_2(j), S_3(j), S_4(j)$.

(1) For $0 \leqslant j < m$, $S_1(j) = S(j)$. This contribution comes from the term x^j of the product $(P_\alpha \cdot P_{\alpha^i})$.
(2) For $0 \leqslant j < m/2$, $S_2(j) = S(j+m)$. This contribution comes from the term x^τ, where $m \leqslant \tau < 3m/2$, of the product $(P_\alpha \cdot P_{\alpha^i})$.

(3) For $m/2 \leqslant j < m$, $S_3(j) = S(j + m/2)$. This contribution also comes from the term x^τ, where $m \leqslant \tau < 3m/2$, of the product $(P_\alpha \cdot P_{\alpha^i})$.
(4) For $0 \leqslant j < m/2$, $S_4(j) = S(j + 3m/2)$. This sum equals the coefficient of the term $x^{m+m/2+j}$ of the product $(P_\alpha \cdot P_{\alpha^i})$, which is equal, $\bmod x^m + x^{m/2} + 1$, to x^j.

Clearly, each of these sums can be evaluated in space $O(\log m)$.

Now, we may assume that the input is legitimate, that is, all the powers of α are correctly presented. Testing whether $\langle G, s, t, 1^m, \alpha, \beta\rangle \in B$ requires computing polynomially many predicates W_{uv}. As noted in Property (4) of the witness function W, there is a nondeterministic logspace machine M_W that computes W_{uv} in the following strong sense: every computation either outputs the correct value of W_{uv} or aborts in a "DON'T KNOW" state, and at least one computation is guaranteed to output the correct value of W_{uv}.

Using this, we will build the nondeterministic $O(\log n + \log m)$ space-bounded machine N that accepts B as follows: Since the elements of the field $GF(2^m)$ have m-bit representations, the machine N cannot write down entries of the field explicitly in its workspace during the computation to check $\sum_{i=0}^{k-1} \alpha^i W_{u_i v_i} = \beta$. Instead, it maintains a $(\log m)$-bit counter that checks, bit by bit, if the above equality holds. To check the equality of the jth bit of $\sum_{i=0}^{k-1} \alpha^i W_{u_i v_i}$ and β, the machine N proceeds as follows: N first initializes a bit $b_j = 0$. Then, sequentially and nondeterministically N computes $W_{u_i v_i}$ for each edge (u_i, v_i) using the machine M_W as a subroutine. If $W_{u_i v_i} = 0$, it goes on to compute the next value $W_{u_{i+1} v_{i+1}}$. If $W_{u_i v_i} = 1$, then it finds the jth bit $(\alpha^i)_j$ of α^i (which is present in the input), and updates $b_j = b_j \oplus (\alpha^i)_j$. Notice that, by design, every computation path of N either computes $W_{u_i v_i}$ correctly (with a "certificate") and proceeds, or it aborts. Finally, N accepts $\langle G, s, t, 1^m, \alpha, \beta\rangle$ if and only if for all j, the jth bits of $\sum_{i=0}^{k-1} \alpha^i W_{u_i v_i}$ and β match.

This completes the proof of $B \in \mathrm{NL}$.

3.3. The TC^0 algorithm for DAG-STCON

By hypothesis, $B \leqslant_m S$. Let f denote the function that reduces B to S. We will show how to solve *DAG-STCON* using f as an oracle. Fix $G = (V, E)$, s and t. Let $n = |V|$ and $k = \binom{n}{2}$. Clearly, $|\langle G, s, t, 1^m, \alpha, \beta\rangle|$ is bounded polynomially in n and m. If f is a TC^0 (or NC^1 or logspace) computable function that reduces B to S, the bound on the length of queries made by f on inputs of length $|\langle G, s, t, 1^m, \alpha, \beta\rangle|$ is some polynomial $q(n, m)$. Let $p(n, m)$ be a polynomial that bounds the number of strings in S of length at most $q(n, m)$. We will choose the smallest m of the form $2 \cdot 3^\ell$ such that $2^m / p(n, m) \geqslant k = \binom{n}{2}$. It is clear that $m = O(\log n)$, and the exact value of m can be easily computed in logspace. Let $\mathbf{F}$ denote the finite extension $GF(2^m)$ of $GF(2)$.

Facts: We first collect some facts about implementing the basic operations of $\mathbf{F}$. The complexity of these operations is important in determining the size, depth, type, and the uniformity of the circuits that we build.

(1) The sum of two elements $\alpha, \beta \in \mathbf{F}$ is just the bitwise exclusive-or of the representations of α and β. The sum of $\ell = n^{O(1)}$-many elements of $\mathbf{F}$ can be computed

by computing in parallel the m bits of the sum, each of which is the parity of ℓ bits. To compute each bit of the sum, we use the fact that the parity can be computed by a TC^0 circuit [7]. The circuitry to perform these additions is also logspace-uniform.

(2) Finding a primitive element ω that generates the multiplicative group $\mathbf{F}^*$ of $\mathbf{F}$ can be done in logspace by exhaustive search. An element $\omega \in \mathbf{F}$ generates $\mathbf{F}^*$ iff the condition "$(\forall \alpha \in \mathbf{F})\,(\exists i < 2^m)\,[\omega^i = \alpha]$" holds. The latter condition can be tested using $O(m) = O(\log n)$ space by maintaining two counters, one that runs through all elements α of $\mathbf{F}$, and another for the exponent i, and doing the multiplications in the straightforward way using $O(m)$ space. Note that in our algorithm for *DAG-STCON*, finding a primitive element is part of the precomputation, and does not have to be implemented in TC^0.

(3) Raising the generator ω to any power $i < 2^m$, as well as computing the discrete logarithm of any element with respect to ω, can be done by AC^0 circuits that hardwire the conjunctive or disjunctive normal form formulas for each bit of the output. It is clear that the formulas themselves can be precomputed using $O(\log n)$ space.

(4) Multiplying $k = n^{O(1)}$ elements of $\mathbf{F}$ can be done by a TC^0 circuit in the following way. Given (wlog. nonzero) elements $\alpha_1, \alpha_2, \ldots, \alpha_k$, first the discrete logarithms $\ell_1, \ell_2, \ldots, \ell_k$ of the k elements are computed with respect to the generator ω. By Fact (3), this can be done by an AC^0 circuit. The next task is to add the k $O(\log n)$-bit integers $\ell_1, \ell_2, \ldots, \ell_k$, and reduce the sum modulo $2^m - 1$. The addition can be done by a logspace-uniform TC^0 circuit (see [5] for details). Since the sum of the k integers is at most $k2^m = n^{O(1)}$, reducing the sum modulo $2^m - 1$ can be easily accomplished by an AC^0 circuit that hardwires the conjunctive or disjunctive normal form formulas for each bit of the output. It is also clear that the circuit description can be precomputed in space $O(\log n)$. Finally, converting the discrete logarithm into the corresponding field element can be done by an AC^0 circuit by Fact (3).

We now describe our parallel algorithm for *DAG-STCON*. Specifically, we describe the structure of the circuit C_n that will solve instances of *DAG-STCON* on n-vertex graphs. To establish the uniformity of the circuit family $\{C_n\}$, we first note that on input 1^n, the appropriate value of m is $O(\log n)$, and can be easily computed using only $O(\log n)$ space (as noted earlier). Thus, in the description of the circuit C_n, $m = O(\log n)$ is a fixed value, and we will express the complexity of various tasks only as a function of n (unlike the case of the language B in Section 3.2, where we needed to show that B can be recognized in space $O(\log n + \log m)$). Furthermore, as we shall see, the circuit C_n will use the results of various pre-computations in the field $\mathbf{F}$; we will appeal to the facts listed above to establish the uniformity of all these operations.

Our parallel algorithm first computes $f(\langle G, s, t, 1^m, \alpha, \alpha^2, \ldots, \alpha^{k-1}, \beta\rangle)$ for all $\alpha, \beta \in \mathbf{F}$, where f is the assumed reduction from B to a sparse set S. (Setting up the required powers of α is an easily accomplished task, since it can be precomputed off-

line in logspace; see Facts above.) For every $\alpha \in \mathbf{F}$, there is a unique element $\beta_\alpha \in \mathbf{F}$ such that $\langle G, s, t, 1^m, \alpha, \beta_\alpha \rangle \in B$, and therefore f maps precisely one tuple of the form $\langle G, s, t, 1^m, \alpha, \beta \rangle$ into S. Since $2^m / p(n,m) \geqslant k$, there is at least one string $z^* \in S$ such that the number of α satisfying $f(\langle G, s, t, 1^m, \alpha, \beta_\alpha \rangle) = z^*$ is at least k. Strings z that have $\geqslant k$ pre-images α will be called *popular*. It is not hard to see that the popular strings can be identified by a TC^0 circuit that counts, for each output z produced by f, the number of (α, β) pairs for which $f(\langle G, s, t, 1^m, \alpha, \beta \rangle) = z$.

For any z, whenever $f(\langle G, s, t, 1^m, \alpha, \beta \rangle) = z$, under the *assumption* that $z \in S$ we have an equation

$$1 W_{u_0 v_0} + \alpha W_{u_1 v_1} + \alpha^2 W_{u_2 v_2} + \cdots + \alpha^{k-1} W_{u_{k-1} v_{k-1}} = \beta$$

in the variables W_{uv}. Thus for every *popular* z, we will have a *system* of at least k such equations; moreover, the system of equations is *correct* if and only if $z \in S$. Of course, there could be many popular z, and we do not know which ones are in S. To handle this, the algorithm will *assume* that every popular z is a string in S, and attempt to solve for the W_{uv}'s for all $u, v \in V$, $u < v$. This scheme produces a polynomial number of sets of solutions. The crucial point is that as long as there is at least one popular $z^* \in S$, one of the assumptions must be correct, and at least one solution produced gives the correct values of the variables W_{uv}.

Assuming the (polynomial number of) candidate solutions to the variables W_{uv} are computed, the formula $\varphi(W)$ described in Section 3.1 may be used to check the validity of each candidate solution. As remarked in Section 3.1, the formula $\varphi(W)$ can be implemented as an AC^0 circuit. Using this formula, all the incorrect solutions will be weeded out, and the correct solution W_{uv} will be obtained. Finally, by computing $W_s \doteq \bigvee_{v \in V} W_{sv}$, it can be determined if $\langle G, s, t \rangle \in DAG - STCON$.

For every popular z, when the equations produced are written as a matrix–vector product of the form $Aw = B$, the $k \times k$ matrix A obtained is a *Vandermonde* matrix. Moreover, since the α's are distinct, the matrix A has full rank over $\mathbf{F}$. It remains, therefore, to show how to solve such systems of equations by a TC^0 circuit, which we describe in the next lemma. The algorithm to be described for solving Vandermonde systems of linear equations appeared in [4] in the resolution of Hartmanis' conjecture for P. (See also [6, 17].) Here we show that it can, in fact, be accomplished in TC^0. Modulo the proof of the lemma, the proof of Theorem 1 is complete.

Lemma 2. *Let* $\mathbf{F} = GF(2^m)$, *where* $m = \mathrm{O}(\log n)$, *and* m *is of the form* $2 \cdot 3^\ell$ *for some integer* $\ell \geqslant 0$. *Solving a system* $Aw = B$ *of* $k = n^{\mathrm{O}(1)}$ *equations in* k *unknowns over the field* $\mathbf{F}$, *where* A *is a* $k \times k$ *Vandermonde matrix of full rank over* $\mathbf{F}$, *can be done by an* $\mathrm{O}(\log n)$*-space uniform circuit of constant depth,* $n^{\mathrm{O}(1)}$ *size, that uses unary NOT gates and unbounded fanin AND, OR, and MAJORITY gates.*

Proof. Observe that an equation of the form $\sum_{j=0}^{k-1} w_j a^j = b$ can be viewed as specifying the value of the polynomial $G(a) \doteq \sum_{j=0}^{k-1} w_j a^j$ at the point $a \in \mathbf{F}$. With this viewpoint, our task is to infer the polynomial G, that is, to find the coefficients w_j

of G. Clearly, if we can evaluate $G(a)$ at n distinct points $a_1,\ldots,a_k \in \mathbf{F}$, then we can recover the coefficients w_j by Lagrange interpolation as follows:

$$G(a) = \sum_{i=1}^{k} G(a_i)Q_i = \sum_{i=1}^{k} b_i Q_i,$$

where

$$Q_i = \frac{(a-a_1)\ldots(a-a_{i-1})(a-a_{i+1})\ldots(a-a_k)}{(a_i-a_1)\ldots(a_i-a_{i-1})(a_i-a_{i+1})\ldots(a_i-a_k)} = \prod_{\ell\neq i}\frac{(a-a_\ell)}{(a_i-a_\ell)}.$$

For $0\leqslant j<k$, w_j is the coefficient of a^j in $G(a)$. Collecting the terms corresponding to a^j, we have

$$w_j = \sum_{i=1}^{k}(-1)^{i+1}\frac{b_i}{\prod_{\ell\neq i}(a_\ell - a_i)}P_{k-j-1}(a_1,\ldots,\hat{a}_i,\ldots,a_k).$$

Here $\hat{a}_i$ denotes that a_i is missing from the list $a_1,\ldots,a_n$, and P_ℓ denotes the ℓth elementary symmetric polynomial, defined as follows:

$$P_0(y_1,\ldots,y_q)=1; \qquad P_\ell(y_1,\ldots,y_q) = \sum_{\substack{I\subseteq[q]\\ |I|=\ell}}\prod_{i\in I} y_i, \quad \ell>0.$$

Using the Facts about computation in $\mathbf{F}$, it is easy to see that computing $b_i/(\prod_{\ell\neq i}(a_\ell - a_i))$ in TC^0 is fairly straightforward. Hence, it suffices to show how to compute the polynomials $P_\ell(a_1,\ldots,\hat{a}_i,\ldots,a_k)$, in logspace-uniform TC^0.

It is easy to see that for $y_1,\ldots,y_q\in\mathbf{F}$, $P_\ell(y_1,\ldots,y_q)$ equals $P_\ell(y_1,y_2,\ldots,y_q,\ 0,0,\ldots,\ 0)$ for any number of extra zeroes. Let $r=|\mathbf{F}^*|$, the number of elements in the multiplicative group of F. We will give an TC^0 algorithm to compute the elementary symmetric polynomial of r elements, not necessarily distinct, from the finite field $\mathbf{F}$. By appending $r-q$ zeroes, we can then compute $P_\ell(y_1,y_2,\ldots,y_q)$.

For $0<\ell\leqslant r$, the value of the elementary symmetric polynomial $P_\ell(y_1,y_2,\ldots,y_r)$ is the coefficient of $X^{r-\ell}$ in $h(X)\doteq\prod_{i=1}^{r}(X+y_i)$. Note that, given any $\alpha\in\mathbf{F}$, $h(\alpha)$ can be evaluated in TC^0, by Facts (1) and (4).

If we write $h(X)$ as $\sum_{i=0}^{r-1}u_iX^i$, the coefficient $u_i=P_{r-i}(y_1,\ldots,y_r)$ for $0\leqslant i<r$. The idea now is to choose α's carefully from $\mathbf{F}$, compute $h(\alpha)$ and compute the coefficients u_i by interpolation. If we choose ω to be a primitive element of order r in $\mathbf{F}^*$, the powers of ω, namely $1=\omega^0,\omega^1,\omega^2,\ldots,\omega^{r-1}$, run through the elements of $\mathbf{F}^*$. For $0\leqslant i<r$, let $v_i=h(\omega^i)$. The relationship between the pointwise values (v_i's) and the coefficients (u_i's) of $h(X)$ can be written as

$$\begin{pmatrix} v_0\\ v_1\\ \vdots\\ v_{r-1}\end{pmatrix} = \begin{pmatrix} 1 & \omega^0 & \omega^{0\cdot 2} & \ldots & \omega^{0\cdot(r-1)}\\ 1 & \omega^1 & \omega^{1\cdot 2} & \ldots & \omega^{1\cdot(r-1)}\\ \vdots & \vdots & \vdots & & \vdots\\ 1 & \omega^{r-1} & \omega^{(r-1)\cdot 2} & \ldots & \omega^{(r-1)\cdot(r-1)}\end{pmatrix}\begin{pmatrix} u_0\\ u_1\\ \vdots\\ u_{r-1}\end{pmatrix}.$$

The above matrix, which we will denote by Ω, is the Discrete Fourier Transform matrix, and is a Vandermonde matrix. Since the powers of ω are all distinct, Ω is invertible, and one can compute the coefficients u_i by $(u_0,\ldots,u_{r-1})^{\mathrm{T}} = \Omega^{-1}(v_0,\ldots,v_{r-1})^{\mathrm{T}}$. The crucial advantage over the earlier Vandermonde system is that with this particular choice of Ω, the matrix Ω^{-1} has a simple explicit form: the (i,j)th entry of Ω^{-1} is just $\omega^{-(i-1)(j-1)}$. Computing the coefficients of $h(X)$ is now simply a matrix–vector multiplication, easily accomplished in TC^0. This completes the proof of the lemma. □

As a corollary of the proof of Theorem 1, we obtain the following:

Corollary 3 (Conjecture of J. Hartmanis [8]). *There is a sparse hard set for* NL *under logspace many-one reductions if and only if* NL = Logspace.

The proofs of the next two theorems combine the above technique with ideas from Cai et al. [2] and from Cai et al. [2], Van Melkebeek [14], respectively.

Theorem 4. *If there is a sparse hard set for* NL *under logspace-computable randomized many-one reductions with two-sided error, then* NL = RL, *where* RL *is the class of languages accepted by logspace Turing machines with two-way access to the random tape. If there is a sparse hard set for* NL *under randomized many-one reductions with two-sided error that are computable in logspace-uniform* NC^1, *then* $\mathrm{NL} \subseteq \mathrm{RNC}^1$.

Theorem 5. *If there is a sparse hard set for* NL *under logspace bounded truth-table reductions, then* NL = L. *If there is a sparse hard set for* NL *under bounded truth-table reductions computable in logspace-uniform* NC^1, *then* NL *equals logspace-uniform* NC^1.

References

[1] L. Berman, J. Hartmanis, On isomorphisms and density of NP and other complete sets, SIAM J. Comput. 6 (1977) 305–321. A preliminary version appeared in STOC 1976.
[2] J. Cai, A. Naik, D. Sivakumar, On the existence of hard sparse sets under weak reductions, Proc. 13th Annual Symp. on Theoretical Aspects of Computer Science (STACS), 1996, pp. 307–318.
[3] J. Cai, M. Ogihara, Sparse sets versus complexity classes, in: L. Hemaspaandra, A. Selman (Eds.), Complexity Theory Retrospective II, Springer, Berlin, 1997.
[4] J. Cai, D. Sivakumar, The resolution of a Hartmanis conjecture, Proc. 36th Annual IEEE Symp. on Foundations of Computer Science, 1995, pp. 362–373. To appear in the FOCS'95 special issue of the Journal of Computer and System Sciences.
[5] A. Chandra, L. Stockmeyer, U. Vishkin, Constant-depth reducibility, SIAM J. Comput. 13 (1984) 423–439.
[6] W. Eberly, Very fast parallel polynomial arithmetic, SIAM J. Comput. 18(5) (1989) 205–217.
[7] M. Furst, J. Saxe, M. Sipser, Parity, circuits, and the polynomial-time hierarchy, Math. Systems Theory 17 (1984) 13–27.
[8] J. Hartmanis, On log-tape isomorphisms of complete sets, Theoret. Comput. Sci. 7(3) (1978) 273–286.
[9] L. Hemachandra, M. Ogiwara, O. Watanabe, How hard are sparse sets?, Proc. 7th Annual IEEE Conf. on Structure in Complexity Theory, 1992, pp. 222–238.

[10] N. Immerman, Languages that capture complexity classes, SIAM J. Comput. 16(4)(1987) 760–778.
[11] N. Immerman, Nondeterministic space is closed under complementation, SIAM J. Comput. 17 (1988) 935–938.
[12] R. Karp, R. Lipton, Turing machines that take advice, L'enseignement Math. 28(3/4) (1982) 191–209.
[13] S. Mahaney, Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis, J. Comput. System Sci. 25(2) (1982) 130–143.
[14] D. Van Melkebeek, Reducing P to a sparse set using a constant number of queries collapses P to L Proc. 11th IEEE Conf. on Computational Complexity, 1996, pp. 88–96.
[15] M. Ogihara, Sparse hard sets for P yield space-efficient algorithms, Proc. 36th Annual IEEE Sympo. on Foundations of Computer Science, 1995, pp. 354–361.
[16] M. Ogiwara, O. Watanabe, On polynomial-time bounded truth-table reducibility of NP sets to sparse sets, SIAM J. Comput. 20(3) (1991) 471–483. A preliminary version appeared in STOC 1990.
[17] F. Preparata, Inverting a Vandermonde matrix in minimum parallel time, Inform. Process. Lett. 38 (1991) 291–294.
[18] W. Savitch, Maze recognizing automata and nondeterministic tape complexity, J. Comput. System Sci. 7 (1973) 389–403.
[19] R. Soare, Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets, Springer, Berlin, 1987.
[20] R. Szelepcsényi, The method of forcing for nondeterministic automata, Bulle. EATCS 33(1987) 96–100.
[21] J.H. van Lint, Introduction to Coding Theory, Springer, Berlin, 1991.
[22] P. Young, How reductions to sparse sets collapse the polynomial-time hierarchy: a primer (Part I), SIGACT News 23(3) (1992) 107–117.
[23] P. Young, How reductions to sparse sets collapse the polynomial-time hierarchy: a primer (Part II), SIGACT News 23(4) (1992) 83–94.